

[0016] FIG. 6 illustrates the use of the local variable pointers upon returning from a method; and

[0017] FIG. 7 illustrates a preferred embodiment of cache-based data storage (including a “RAMset”) in the JSM of FIG. 3;

[0018] FIG. 8 illustrates the mapping of a contiguous block of main memory onto a RAMset;

[0019] FIG. 9 illustrates another mapping of a contiguous block of main memory onto a RAMset;

[0020] FIGS. 10 and 11 illustrate the operation of the RAMset in an overflow condition;

[0021] FIG. 12 illustrates the operation of the RAMset in an underflow condition;

[0022] FIG. 13 illustrates a RAMset split into two portions in accordance with a preferred embodiment of the invention; and

[0023] FIG. 14 shows various states and the transitions between states of the RAMset in accordance with embodiments of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0024] The following discussion is directed to various embodiments of the invention. Although one or more of these embodiments may be preferred, the embodiments disclosed should not be interpreted, or otherwise used, as limiting the scope of the disclosure, including the claims, unless otherwise specified. In addition, one skilled in the art will understand that the following description has broad application, and the discussion of any embodiment is meant only to be exemplary of that embodiment, and not intended to intimate that the scope of the disclosure, including the claims, is limited to that embodiment.

[0025] The subject matter disclosed herein is directed to a programmable electronic device such as a processor having memory in which “local variables” associated with a stack-based language (e.g., Java) and pointers associated with the local variables may be stored. The term “local variables” refers to temporary variables used by a method that executes on the processor. Multiple methods may run on the processor and each method preferably has its own set of local variables. In general, local variables have meaning only while their associated method is running. The stack-based language may comprise Java Bytecodes although this disclosure is not so limited. In Java Bytecodes, the notion of local variables (“LVs”) is equivalent to automatic variables in other programming languages (e.g., “C”) and other termed variables in still other programming languages. This disclosure, however, is not limited to Java, Java methods, and Java local variables. The principles disclosed below are applicable to any system that manages a stack and includes “put block” and “pop block” operations to push a block of data onto a stack or pop a block of data from a stack.

[0026] The following describes the operation of a preferred embodiment of such a processor in which the methods and local variables may run and be used. Other processor architectures and embodiments may be used and thus this disclosure and the claims which follow are not limited to any particular type of processor.

[0027] The processor described herein is particularly suited for executing Java™ Bytecodes, or comparable code. As is well known, Java is particularly suited for embedded applications. Java is a relatively “dense” language meaning that on average each instruction may perform a large number of functions compared to various other programming languages. The dense nature of Java is of particular benefit for portable, battery-operated devices that preferably include as little memory as possible to save space and power. The reason, however, for executing Java code is not material to this disclosure or the claims that follow.

[0028] Referring now to FIG. 1, a system 100 is shown in accordance with a preferred embodiment of the invention. As shown, the system includes at least two processors 102 and 104. Processor 102 is referred to for purposes of this disclosure as a Java Stack Machine (“JSM”) and processor 104 may be referred to as a Main Processor Unit (“MPU”). System 100 may also include memory 106 coupled to both the JSM 102 and MPU 104 and thus accessible by both processors. At least a portion of the memory 106 may be shared by both processors meaning that both processors may access the same shared memory locations. Further, if desired, a portion of the memory 106 may be designated as private to one processor or the other. System 100 also includes a Java Virtual Machine (“JVM”) 108, compiler 110, and a display 114. The JSM 102 preferably includes an interface to one or more input/output (“I/O”) devices such as a keypad to permit a user to control various aspects of the system 100. In addition, data streams may be received from the I/O space into the JSM 102 to be processed by the JSM 102. Other components (not specifically shown) may be included as desired. As such, while system 100 may be representative of, or adapted to, a wide variety of electronic systems, an exemplary electronic system may comprise a battery-operated, mobile cell phone such as that shown in FIG. 2. As shown in FIG. 2, a mobile communications device includes an integrated keypad 412 and display 414. The JSM 102 and MPU 104 noted above and other components may be included in electronics package 410 which may be coupled to keypad 410, display 414, and a communications transceiver (e.g., radio frequency (“RF”) circuitry) 416 which may be connected to an antenna 418.

[0029] Referring again to FIG. 1, as is generally well known, Java code comprises a plurality of “Bytecodes” 112. Bytecodes 112 may be provided to the JVM 108, compiled by compiler 110 and provided to the JSM 102 and/or MPU 104 for execution therein. In accordance with a preferred embodiment of the invention, the JSM 102 may execute at least some, and generally most, of the Java Bytecodes. When appropriate, however, the JSM 102 may request the MPU 104 to execute one or more Java Bytecodes not executed or executable by the JSM 102. In addition to executing Java Bytecodes, the MPU 104 also may execute non-Java instructions. The MPU 104 also hosts an operating system (“O/S”) (not specifically shown), which performs various functions including system memory management, the system task management that schedules the JVM 108 and most or all other native tasks running on the system, management of the display 114, receiving input from input devices, etc. Without limitation, Java code may be used to perform any one of a variety of applications including multimedia, games or web based applications in the system 100, while non-Java code, which may comprise the O/S and other native applications, may still run on the system on the MPU 104.